DOI:

嵌入式操作系统内存管理机制及内存保护分析

哈渭涛[1,2], 罗 琪[1,2]
(1.西安电子科技大学, 西安 710071; 2.渭南师范学院 计算机科学系, 陕西 渭南 714000)

摘 要: 文章介绍了嵌入式操作系统内存管理的基本原理及目前的管理方式存在的问题, 并在此基础上介绍了一种嵌入式系统基于页机制的内存保护方法。

关键词: 嵌入式操作系统; 内存管理; 内存保护
中图分类号: TP333.1 文献标志码: A 文章编号: 1009-5128(2007)02-0063-03
收稿日期: 2006-04-27
基金项目: 渭南师范学院科研基金项目(06YKS031)
作者简介: 哈渭涛(1975-), 男(回), 陕西渭南人, 西安电子科技大学硕士研究生, 渭南师范学院计算机科学系讲师.

嵌入式系统是以应用为中心, 计算机技术为基础, 软硬件可裁剪, 适应于特定应用系统, 对功能、可靠性、成本、体积、功耗有严格要求的计算机系统. 嵌入式系统应用相当广泛, 从日常生活中的电子产品如数码相机、DVD播放器到航空航天器的导弹制导系统、飞行控制系统等, 都可以看到嵌入式系统的影子.

1 嵌入式系统内存管理存在的问题

内存管理机制是嵌入式系统研究中的一个重点和难点的问题. 针对有内存管理单元(MMU)的处理器设计的一些桌面操作系统(如 Windows, Linux)都使用了虚拟存储器的概念, 虚拟内存地址被送到MMU, 在这里, 虚拟地址被映射为物理地址, 实际存储器被分割为相同大小的页面, 采用分页的方式载入进程. 一个程序在运行之前, 没有必要全部装入内存, 而是仅将那些当前要运行的部分页面装入内存运行. 大多数嵌入式系统是针对没有MMU的处理器设计的, 因此不能使用处理器的虚拟内存管理技术, 而采用实存储管理策略, 从而对内存的访问是直接的. 它对地址的访问不需要经过MMU, 而是直接送到地址线上输出, 所有程序中访问的地址都是实际的物理地址. 而且, 大多数嵌入式操作系统对内存没有保护, 各个进程实际上共享一个运行空间. 一个进程在执行前, 系统必须为它分配足够的连续地址空间, 然后全部载入主存储的连续空间. 这样对嵌入式系统的内存管理和保护就显得极为重要.

2 嵌入式系统内存管理的基本方案

嵌入式系统中最基本的内存管理方案有两种——静态分配和动态分配.

一般的嵌入式系统都支持静态分配, 因为像中断向量表、操作系统映像这类的程序段, 其程序大小在编译和链接时是可以确定的. 而是否支持动态分配主要基于两个方面的考虑: 首先是实时性和可靠性的要求, 其次是成本的要求. 对于实时性和可靠性要求极高的系统(硬实时系统), 不允许延时或者分配失效, 必须采用静态内存分配. 除了基于成本的考虑外, 用于汽车电子和工业自动化领域的一些系统也没有动态内存分配. 然而, 仅仅采用静态分配, 使系统失去了灵活性. 必须在设计阶段就预先知道所需要的内存并对之做出分配; 必须在设计阶段就预先考虑到所有可能的情况, 因为一旦出现没有考虑到的情况, 正在运行的系统就无法处理. 这样的分配方案必然导致很大的浪费.

虽然动态内存分配会导致响应和执行时间不确定、内存碎片等问题, 但是它的实现机制灵活, 给程序实现带来极大的方便, 有的应用环境中动态内存分配甚至最必不可少的. 比如, 嵌入式系统中使用的网络协议栈, 在特定的平台下, 为了比较灵活地调整系统的功能, 在系统中各个功能之间作出权衡, 必须支持动态内存分配.

3 嵌入式系统内存管理中的存储保护

在动态内存分配方案中内存保护就显得极为重要, 下面介绍一种基于页式管理机制下的内存保护措

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

施. 设定系统的名地址空间和实地址空间具有相同的大小, 所以任何逻辑地址和它相对应的物理地址具有相同的值, 因为所有的逻辑页都在内存中, 所以无需请页调度机制. 采用这种设计实现起来非常简单, 而且可以避免传统的页机制对系统效率的影响.

内存的保护功能是建立在域(realm)的基础之上. 内核和用户进程都有自己的域, 一个域是一组内存页, 它们由一个或多个页表索引组成, 具有相同的访问权限. 由与内核相关的页面组成的域被称为内核域(KernelMap), 内核域包括指向其它任何的域的指针, 但是它只能被核心态访问.

系统中使用了 4个基本的内存管理数据结构: (1) Region Table, 跟踪系统中物理内存的分配情况. (2) Page Table(页表): 每个域都是由一个或多个页表索引的页面组成, 这些页表记录了与该域相关的各种安全参数. (3) Realm(域): 与进程相关, 记录该进程所占有的所有内存页. (4) KernelMap(内核域): 与内核相关的页面组成的域包括指向其它任何的域的指针.

(1) Region 与 Region Table

从系统中申请得到的一块连续的内存区域被称为一个 region, 系统中内存的分配与释放是以页为单位的, 所以每个 region 都是页对齐的. 一个 region的数据结构如图 1所示, start记录该 region的起始地址, Len记录其大小(以页面数为单位), procid记录该 region所属进程的进程号(向系统申请该 region 的线程所在的进程). 同一进程内的所有线程都可以访问该 region 通过 prev和 next两个指针可以将多个 region 合起来建立一个双向链表.

一个 Region Table就是由多个 region 所构成的双向链表, 它按各个 region 的 start值的大小排序. 整个系统中共有两个 Region Table, 一个 free Region Table 用于保存尚未分配或已被释放的 region, 另一个 allocated Region Table 用于保存已被分配给各个进程的 region.

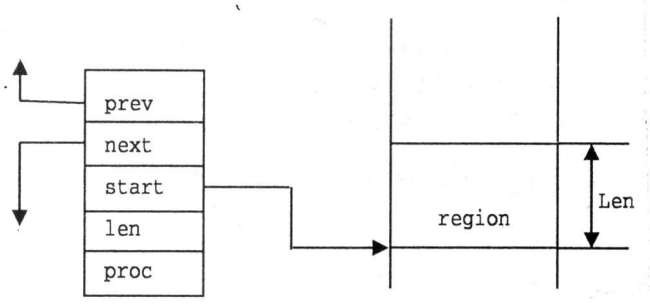

图 1 一个 region数据结构

与 region相关的各项操作:

region_init(&region)

region_insert(region, list)

region_remove(region, list)

region_split(region, size)

region_free(region)

(2) Page Tables

采用分页机制后, 程序中使用的是虚拟地址, 其地址空间被分成了固定大小的页, 每页 4B. 而访问物理内存, 需要使用物理地址. 页目录和页表就是完成将虚地址映射到物理地址上的工作. 如图 2所示, 一个页目录大小为 4kB, 放在一个物理页中. 由 1024个 4B字节的页目录项组成. 页目录项的大小为 4B, 所以一个页目录中有 1024个页目录项. 页目录中的每一项的内容(每项 4B)高 20b用来放一个页表(页表放在一个物理页中)的物理地址, 低 12b放着一些标志. 一个页表的大小为 4kB, 放在一个物理页中. 由 1024个 4B的页表项组成. 页表项的大小为 4B, 所以一个页表中有 1024个页表项. 页表中的每一项的内容(每项 4B)高 20b用来放一个物理页的物理地址, 低 12b放着一些标志. 由于虚拟地址空间和实地址空间一样大, 因此由页目录和页表管理的虚拟地址空间的大小就是实际应用系统的物理内存大小. 以一个 64MB系统为例, 一个页表可映射 4MB物理地址, 共需要 1个页目录和 16个页表.

U/S: 用户级/特权级 R/W: 读/写 Avail: 其它用途(如页的调度)

图 2 页目录和页表中每个入口的结构

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

系统以页为最基本的分配单位, 同时也是最基本的内存保护单位. 如图 2, 每个页表项指向一个物理页, 可以在低 12b的标志位上设定其安全特性, 如读 写, 用户级还是特权级. 通过为内核和每个进程所属的页设定不同的安全特性, 可以有效地实现内存保护, 防止越权访问.

(3) Realm 与 KernelMap

两个 region list维护整个系统的内存使用情况. 对于每个进程还需要一个结构来记录它自己所获得的内存空间, 这个结构被称为域. 每个进程维护一个域, 一个域大小为 4kB, 放在一个物理页中. 域中一个入口的大小为 8B. 其中高 32b用来记录该入口指向的 region的首址. 接着的 10b用来记录该 region的大小(以页为单位). 剩余的 bits作为标志位保存与该 region相关的安全特性. 一个域最多可容纳 512个入口, 也就是说一个进程最多可以同时申请 512个 region.

内核用于记录自身内存占用情况的结构是一个特殊的域, 称为内核域(KernelMap), 内核域中每个入口的结构与用户进程中的基本相同, 只是在内核域中除了包含内核自身的内存使用信息外还为每个进程保留了一个入口, 指向该进程的域. 通过这些入口, 内核可以方便地访问各个进程的代码和数据.

与域相关的各项操作:

vm_map(realm, startaddr, pagenum, mode)

vm_unmap(realm, startaddr)

vm_kmap(kmap, startaddr, pagenum, mode)

vm_unkmap(kmap, startaddr)

(4)内存管理实现

内存管理模块为内存的申请释放提供的函数接口: kmalloc()和 kfree()为内核分配释放内存, 由核内程序直接函数调用. malloc()和 free()为核外进程申请分配内存, 由核外应用通过软中断机制调用. malloc(&region, size)算法: 在各个函数中按照一定的规则调用与 region相关的各项操作和与域相关的各项操作, 采用首次适应算法可以使内分配快速简洁. 对 free list中相邻的 region进行整合可以有效防止系统长时间运行后在内存中产生大量的碎片. Region list采用双向链表就是为了方便在释放的时候对相连的 region进行合并. vm_map()和 vm_kmap()在将一个 region映射到一个域的同时还对与该 region相关页的页表项设置标志位, 利用硬件对页机制的支持, 实现内存保护.

4 总结

嵌入式系统由于使用范围的特殊性, 它对内存的要求也就更加地突出, 这就导致内存管理在嵌入式系统中显得尤为重要. 本文介绍的内存管理和保护方法, 是一种简洁、高效且方便移植的方法, 在众多的嵌入式系统中得到应用.

参考文献:

[1] 孙益辉, 陈凯, 白英彩. 嵌入式操作系统内存管理机制分析及改进[J]. 计算机应用与软件, 2006, 23(3): 98-115.

[2] 刘云生, 胡昊明. 嵌入式实时操作系统中基于页的内存保护[J]. 计算机工程, 2005, 31(18): 53-55.

[3] 卢春鹏. 一种嵌入式系统的内存分配方案[J]. 单片机与嵌入式系统应用, 2002, (12): 12-16.

[4] 李娜, 方彦军. 嵌入式系统内存规划方法的研究[J]. 微型机与应用, 2005, (11): 74-76.

[责任编辑　牛怀岗]

On Embedded Operating System Memory Management Mechanism and Memory Protection

HA Wei-tao[1,2], LUO Qi[1,2]

(1. Xidian University, Xi'an 710071, China;

2. Department of Computer Science, Weinan Teachers University, Weinan 714000, China)

Abstract: The paper introduces the basic theory of memory management in embedded operating system and the problem in the way of management. The design of a page-based memory protection in embedded operating system is described.

Key words: embedded operating system; memory management; memory protection

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net